JPL D-73355

Earth Observing System

Multi-angle
Imaging
Spectro-
Radiometer

MISR Science Data Product Guide

Veljko Jovanovic
Kyle Miller
Brian Rheingans
Catherine Moroney

JPL

Jet Propulsion Laboratory
California Institute of Technology

May 7, 2012

JPL D-73355

Multi-angle Imaging SpectroRadiometer (MISR)

MISR Science Data Product Guide

Approval signatures are on file with the MISR Project.
To determine the latest released version of this document, consult the MISR web site (http://misr.jpl.nasa.gov).

David J. Diner
MISR Principal Investigator

Earl Hansen
MISR Project Manager

Jet Propulsion Laboratory
California Institute of Technology

May 7, 2012

The research described in this publication was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Document Change Log

Revision	Date	Affected Portions and Description
	1 May 2012	All, original release

This document applies to all of the MISR science data products.

TABLE OF CONTENTS

1 MISR SCIENCE DATA PRODUCTS

1.1 OVERVIEW

The Multi-angle Imaging SpectroRadiometer (MISR) project is a component of the Earth Observing System (EOS) Terra Mission and the EOS Data Information System (EOSDIS), which in themselves are components of the National Aeronautics and Space Administration's (NASA) Earth Science Enterprise. An integral part of the MISR project is the Science Data Processing (SDP) of the observations coming from the MISR instrument on-board the EOS-TERRA satellite.

MISR SDP exists to produce science and supporting data products from MISR instrument data. All functions of the MISR SDP system are directed toward this goal. MISR SDP does not operate as an independent entity, but rather is linked to the functionality of the EOSDIS at the Langley Research Center (LaRC) Atmospheric Sciences Data Center (ASDC), also known as the LaRC Distributed Active Archive Center (DAAC). The ECS (EOSDIS Core System) ingest subsystem at the LaRC DAAC is the agent for receiving and organizing all of the input data needed by MISR SDP. These data are then made available to MISR SDP through the data server and staging facilities provided at the LaRC DAAC. After MISR standard data processing is complete, the standard output products are archived through the EOSDIS data server and made available to users through ECS client services.

The MISR Science Computing Facility (SCF) at the Jet Propulsion Laboratory (JPL) supports the development of MISR science algorithms and software, instrument calibration and performance assessment, as well as providing quality assessment and data validation services with respect to MISR SDP. The MISR SCF is used to produce software, supporting data, and coefficients that are required to operate MISR SDP software at the LaRC DAAC.

MISR SDP depends upon the availability of MISR instrument data, internal data sets produced at the MISR SCF, and external data sets that are products of other EOS data processing systems. This document is not meant to be the definitive description of the external data sets that are products of other EOS data processing systems and utilized by MISR SDP. For details of the other internal data sets, see the MISR Software Interface Specification (SIS) document for internal interfaces and the Data Management Plan (DMP), which describes MISR SCF activities in regard to archiving data at the MISR SCF.

1.2 SCOPE

The purpose of this document is to describe a template for MISR science data products. The full details of all the other current MISR standard products, as well as the ancillary datasets used in their generation, can be found in the appropriate MISR Data Products Specification documents.

In Section 3 of this document the general file structure for the geophysical and ancillary MISR science data products is described. In particular, MISR SDP dependence on both HDF-EOS swath and grid formats is outlined, together with the HDF and HDF-EOS structures that MISR SDP uses to store metadata.

Section 4 contains a description of the georegistration method including information for obtaining the latitude and longitude of a MISR pixel. Since most MISR data products are registered to the SOM map projection, it is important to understand the conversions in the Section 4 in order to compare MISR products to data from other sources.

1.3 CONTROLLING DOCUMENTS

1) MISR Science Data Processing Functional Requirements Document, (FRD) JPL D-12417, September 1996 (or latest version).
2) MISR Experiment Implementation Plan, Volume III, Science, Data Processing, and Instrument Operations, Technical and Management Plan (EIP), JPL D-11520, 24 January 1996 (or latest version).
3) MISR Science Data System Software Management Plan (SMP), JPL D-11641, February 1996 (or latest version).
4) SDPIO Implementation Handbook, JPL D-16392, January 1999 (or latest version).
5) MISR Data System Science Requirements, JPL D-11398, September 1996 (or latest version).

APPLICABLE DOCUMENTS

6) Science User's Guide and Operations Procedure Handbook for the ECS Project, HAIS 193-205-SE1-001 (or latest version).
7) Interface Requirements Document Between EOSDIS Core System (ECS) and Science Computing Facilities, HAIS 209-CD-005-005, March 1996 (or latest version).
8) EOSDIS Core System Science Information Architecture, HAIS working paper FB9401V2 (or latest version).
9) Software Implementation Guidelines, JPL D-10622 (or latest version).
10) MISR Science Data System Error Policy, JPL D-13137 (or latest version).
11) SDP Toolkit Users Guide for the ECS Project, HAIS 194-809-SD4-001 (or latest version).

2 MISR SCIENCE DATA PROCESSING

2.1 OVERVIEW

Multi-angle Imaging SpectroRadiometer (MISR) science data processing (SDP) at the Langley Research Center (LaRC) Distributed Active Archive Center (DAAC) requires data sets generated internally at the MISR Science Computing Facility (SCF), as well as external data sets generated outside of the MISR project. The internal data sets generated by the MISR SCF are called ancillary data sets.

2.2 ANCILLARY DATA SET GENERATION

The information used to generate the ancillary data sets is listed in **Table 2-1**. The ancillary data sets generated at the MISR SCF, listed in **Table 2-2** are archived at the MISR SCF, sent to the LaRC DAAC to be archived, and then used as an input to MISR SDP at the LaRC DAAC.

2.2.1 MISR SDP Ancillary Data Set Dependencies

Table 2-1: MISR SDP Ancillary Data Set Dependencies

Information Required for the MISR TC_CLOUD Product
Spacecraft Ancillary Data
Predicted Spacecraft Orbit
Radiometric Calibration Reference Imagery
Geometric Calibration Reference Imagery
Global Digital Elevation Model
Solar Irradiance Model
Earth-Sun Ephemeris

2.2.2 MISR SDP Ancillary Data Sets

Table 2-2: MISR SDP Ancillary Data Sets

Product	ESDTs	File Description
Ancillary Radiometric Product	MIANCARP	Radiometric calibration coefficients
Ancillary Geographic Product	MIANCAGP	Geographic information (lat, lon, DEM, etc)
Camera Geometric Model	MISANCGM	Geometric calibration information for the MISR cameras
Projection Parameters	MIANPP	Parameters used to georegister the MISR radiance data

3 MISR PRODUCTS: FILE FORMAT OVERVIEW

3.1 HDF AND HDF-EOS FILE STRUCTURES

This section describes the specifications for the MISR products that will be archived at the NASA LaRC DAAC. The MISR files (with one exception as noted below) are implemented in the Hierarchical Data Format (HDF). Most, but not all, of the MISR standard data products are in one of two file formats: HDF-EOS Swath or HDF-EOS Grid, which are extensions of the original HDF as developed by the National Center for Supercomputing Applications (NCSA). The HDF-EOS file interfaces were developed as part of the EOS Core System (ECS). Standard NCSA HDF terminology, as well as the EOS developed interface terminology, is used in this document when describing these files.

The HDF-EOS data structures used by MISR (Swath and Grid) have been defined within the HDF framework and are supported by special application programming interfaces (API) which aid the data producer and user in writing to and reading from these files. These APIs allow data products to be created and manipulated in ways appropriate to each datatype, without regard to the actual HDF objects and conventions underlying them. MISR Swath products are composed of the data acquired on the illuminated portion of the Earth along one given orbit pass. Most MISR Grid products contain data covering the same geographic range as the Swath products. In these cases, the Grid format allows the data to be represented in a map-projected fashion. The map-projected format is necessary because MISR data product generation schemes often involve the use of data acquired from different cameras. Such use requires data sets to be co-registered, and projecting the data is the most efficient way to represent co-registration. There are other cases in which the HDF-EOS Grid format is used to display data on a global map grid instead of focusing on a narrow “swath area”. MISR Level 3 products are examples of the global use of the HDF-EOS Grid format.

The file specifications given here are in terms of the logical implementation of MISR standard data products in HDF and do not describe the actual physical layout of the files, although there is an attempt to show what the physical layout of a file looks like. The same data object may exist in different relative locations for two iterations of a product file. The locations are determined by HDF on a file-by-file basis and are not important to actually accessing the data using API calls.

3.2 MISR PRODUCT DATA FORMATS

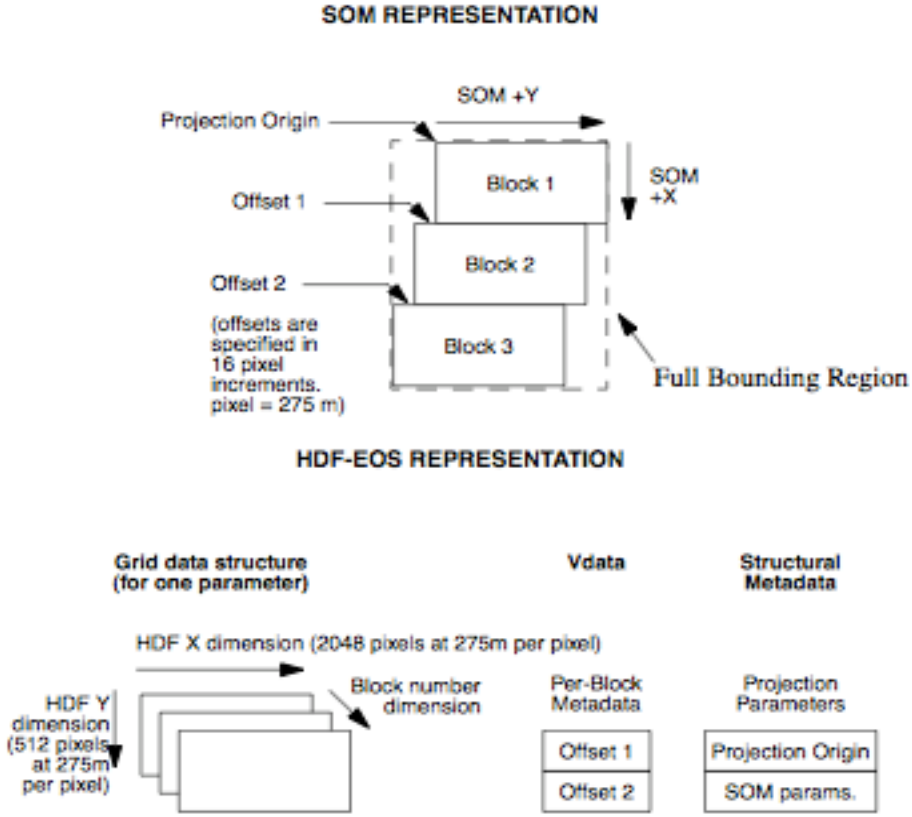
Of all of the MISR standard science data products, only the MISR Ancillary Radiometric Product and Aerosol Climatology Product files use standard NCSA-supplied HDF file structures.

The MISR Level 1B2 and Level 2 products are HDF-EOS Grid file. The storage of map projection parameters is part of the format, and routines to access the data in Grid format by geolocation are supplied in the Grid API. MISR stores “swath-like” products at Level 1B2 and Level 2 in a space-based map projection. In particular, MISR SDP breaks up L1B2 and L2 swaths into equal-sized blocks. The term “block” refers to a pre-defined, static, fixed-size, rectangular SOM (Space-Oblique Mercator) region on the Earth which a) is wide enough to

contain the horizontal overlap of all 9 MISR camera views at low latitudes, b) is the geographic unit over which MISR SDP is attempted and c) is the standard unit of MISR data access. The block construct enables the co-registration of 9 different images with a minimal waste of space and processing effort. Changes were made to the HDF-EOS Grid implementation specific to MISR SDP needs in order to handle these blocks as an additional dimension to a Grid dataset. This implementation is referred to as the "Stacked-block" Grid implementation. The user is referred to Section 4.4, for details about Stacked-block grid data access.

In brief, the solution to meet MISR's needs for its Level 1B2 and Level 2 data products is to "stack" all of the blocks of an orbit into a single dataset, where the "third" dimension for the dataset becomes the block number (Figure 3-1). Groups of parameters of a product can be stored in these "stacked-block" Grid data structures, but each parameter in the structure must have the same x and y dimensions (i.e., same resolution). Within a Grid dataset, parameters can also be grouped into what HDF-EOS calls a "field", but each parameter in the field must be of the same data type (e.g., 2-byte integer).

Figure 3-1 MISR SOM Representation in HDF-EOS



180 blocks are required to cover the seasonal sunlit ground under a single path. Each of the 233 WRS Terra orbit paths has a separate set of 180 MISR blocks defined to span it. The MISR Ancillary Geographic Product (AGP) is the reference dataset containing full lat/lon information about block locations. Refer to the MISR AGP Data Product Specification document for additional information about the AGP (section 9.4 in the previous MISR Data Products Specifications Document (Revision S)). Each block could have been its own grid in a separate

file, but that model did not meet MISR SDP IO constraints. Attempting to make one rectangular grid spanning the entire swath range was size-prohibitive. At product design time, HDF tiling and compression mechanisms could not support that large a region. The compromise solution was to store only the projection origin for Block 1 and save in a separate structure the horizontal offset of each block beneath Block 1. The user should be aware that data acquisition does not always begin in Block 1. However, even subsetted MISR Grid data files always contain HDF structures that begin at Block 1. Empty blocks contain fill data. This offset is specified in integer pixels from the upper left hand corner of Block 1. The storing of these offsets is taken care of by a Grid library routine (GDblkSOMoffset). Offsets may be retrieved with the same routine. With the projection origin information, the projection information, and the block offset information, transformations between (block/line/sample <-> SOM X/Y <-> lat/lon) are possible. HDF-EOS and the accompanying GCTP map projection library do most of the work automatically. The user is again referred to Section 4.

3.3 MISR PRODUCT METADATA FORMATS

No matter what kind of product file is created, metadata must be associated with it for descriptive purposes within the ECS environment. All products have an accompanying simple XML file containing metadata that is produced at the same time. These associated XML files contain what is called the Core, or Inventory, Metadata, which is the second type of metadata described below. For MISR standard data product files that use only native HDF objects, Inventory Metadata is the only type of metadata that is produced for the product.

For MISR HDF-EOS Swath and Grid file types, six classes of metadata may be used: 1) Structural Metadata, 2) Core Metadata, 3) Product Metadata, 4) File Metadata, 5) Grid/Swath Metadata and 6) Per-block Metadata (for Grid files only). The first three classes of metadata are recognized by ECS and can be searched in the ECS Data Server database. The last three classes were invented by MISR and contain values required by MISR processing. Attempts to provide convenient data access often clash with requirements to meet programmatic standards. Since metadata are usually small in size, the MISR team handled such clashes by including redundant information in places where it was warranted.

3.3.1 Structural Metadata

Structural Metadata are written into HDF files automatically by HDF-EOS software when writing out HDF-EOS files. These metadata describe the structure of the file in terms of its dimensions, Swath or Grid characteristics, projection (for Grid only), and data fields. These metadata are used by HDF-EOS software to recognize file structures when reading the data.

3.3.2 Core Metadata

Core, or Inventory, Metadata provide granule level information used for ingesting, cataloging, and searching the data product. These metadata are written into HDF-EOS files by Toolkit metadata calls. A Metadata Configuration File (MCF), which describes Inventory Metadata attributes, is used when creating an HDF file using the Toolkit. An additional XML Inventory Metadata file is produced at file-time creation that provides granule level information. This

XML file has the same name as the HDF or HDF-EOS output file with the extension of .xml. These files are not described in this document.

3.3.3 Product Metadata

Product, or Archive, Metadata provide granule level information that is not used for search purposes, but which are important to be kept with the HDF-EOS file. These metadata are also attached by Toolkit metadata calls during product generation and their attributes are also contained in the MCF file. MISR processing does not currently use Archive Metadata, preferring to create and use the next three types of metadata described below.

3.3.4 File Metadata

File Metadata, as used during MISR data processing, contain MISR-specific information that is common to a whole file. These metadata are stored as global attributes that are attached to the standard NCSA-supplied HDF Scientific Dataset (SD) object. (We found it necessary to use SD object global attributes since the HDF-EOS Grid API does not provide a means of storing global data relevant to an entire file as opposed to a single Grid data set.) These metadata are used to process a file, but they are not intended for search purposes use. MISR is currently using this class of metadata to store such things as additional projection information and product statistics. If a file contains only one Grid or Swath dataset, and Grid/Swath Metadata (section 3.3.5 below) are attached at that level, File Metadata may not be included in the file. The values for a particular attribute must all be of the same type.

3.3.5 Per-grid/Per-swath Metadata

Grid Metadata are internal to HDF-EOS files and are used to provide MISR-specific information unique to an individual Grid or Swath dataset in the file. An example of such metadata is the resolution of the data in a Grid or Swath dataset. In the case of Grid files these are Grid attributes attached using HDF-EOS Grid application calls. The values for a particular attribute must all be the same type.

3.3.6 Per-block Metadata

The Per-block Metadata are internal to the file and are used to provide MISR-specific information unique to an individual block of a Grid dataset. This class of metadata is used only in Grid files. Since the HDF-EOS Grid API does not contain structures for dealing with MISR blocks, these metadata are stored using standard NCSA-supplied HDF Vdata tables within the file. A wrapper was written around the native HDF Vdata interface for reading and writing Per-block Metadata. Because native HDF expects a file id returned from Hopen, the file id returned from GDopen cannot be used when calling native HDF routines. Consequently, an HDF-EOS function, called EHidinfo, has been provided for translating grid file ids to native file ids. The native file id is then used in the Vdata calls. The attributes stored in Per-block Metadata include per-block coordinates, such as L1B2 transform information, and statistics.

4 MISR GEOREGISTRATION INFORMATION

4.1 Introduction

The concept of multi-angle imaging, which is being pioneered by the MISR experiment, is distinct in several ways from traditional nadir-viewing, scene-based multi-spectral imaging. All remote sensing data products are contaminated by BRF effects in the cross-track direction. This fact is ignored by most in the remote sensing community. MISR is the only instrument to date that makes a genuine effort to deal with and benefit from BRF effects. It does so by acquiring imagery from multiple angles in the along-track direction in a short enough period of time to characterize the anisotropy of the surface-atmosphere system. The design of MISR data products was shaped largely by the stringent requirements of the experiment. Users of MISR data have requested clarification about product attributes such as the Space Oblique Mercator (SOM) map projection, the stacked-block HDF-EOS Grid format, and the large size and geographical extent of MISR files. This document provides background information on such topics. It also describes the most precise method for extracting georegistration information directly from MISR files.

There are two ways to determine the latitude and longitude of a pixel within a MISR file. The first method involves reading values directly from an ancillary file called the Ancillary Geographic Product (AGP). Because of data volume considerations, it was not practical to include this redundant information within MISR product files. AGP values are reported at a single resolution. Therefore, interpolation may be required to determine Lat/Lon for a particular desired resolution. The AGP is described in detail in the MISR AGP Data Product Specification document (section 9.4 in the previous MISR Data Products Specifications Document (Revision S))[1]. There is one AGP file for each Terra orbital path. The MISR AGP is available for public distribution at the Langley DAAC.

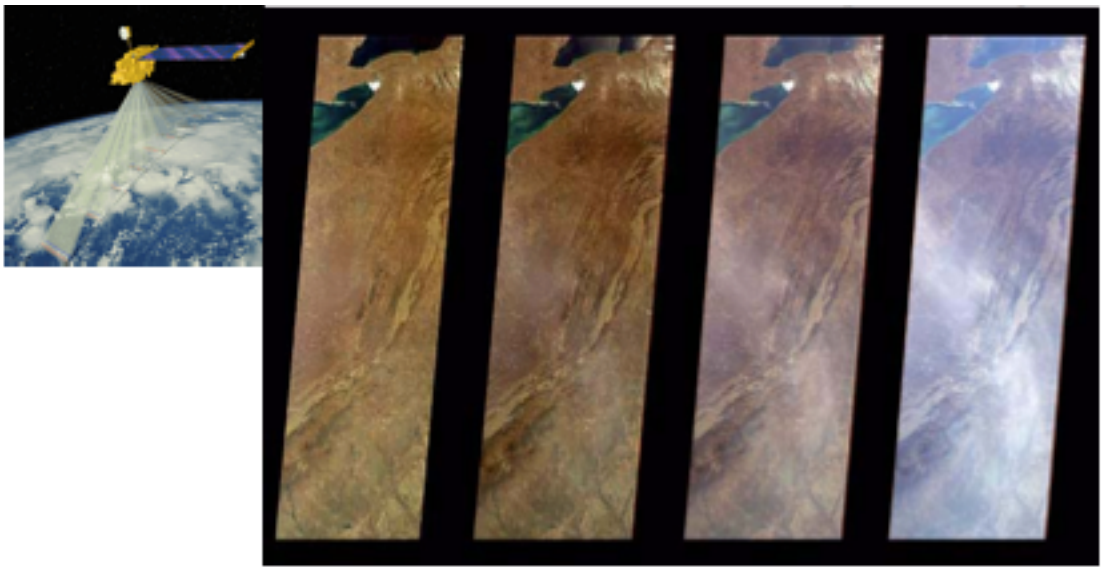
The second method for determining lat/lon is much more flexible and requires no ancillary files. The crux of this operation involves coordinate conversions between MISR (block,line,sample), SOM(X,Y), and Lat/Lon. These conversions are supported by orbital parameters and projection information embedded within all MISR products. Such conversions are readily accomplished using HDF-EOS library access routines and the accompanying GCTP map projections library [3]. More complex operations, such as resampling an entire scene to another map projection, are not difficult once the fundamental conversions described herein are understood.

Individuals who are familiar with the MISR experiment, MISR data products, HDF-EOS, the SOM map projection and the GCTP software library may wish to proceed directly to the sections describing coordinate conversion [4.5]. However, most users will save time by perusing the Background sections first. Information about key Metadata values required for coordinate conversions has been condensed for quick-reference into the Metadata Details section [4.6]. Example C function calls can be found at the end of this document to clarify the function call interfaces that are used in the algorithm described in [4.5].

4.2 Background: The Instrument

The Multi-angle Imaging SpectroRadiometer (MISR) is a new and unique type of satellite instrument. As the Terra satellite moves in its descending polar orbit, each of 9 MISR cameras images the same daylit ground swath, which is 1504 detector samples wide by roughly 70,000 lines long. MISR obtains images for any pixel in the swath from 9 different angles in four different wavelengths. Many remote sensing experiments make use of spectral information to measure physical properties of the Earth's surface, vegetation, atmosphere, and clouds. The novel goal of the MISR design is to make use of angular information as well as spectral. The challenge is that this ambitious scientific task cannot be accomplished unless the 36 pixels MISR obtained for each location can be registered together accurately, 9 cameras x 4 bands = 36. An additional complication in the instrument configuration is that not all 4 bands are acquired at the same resolution in all cameras. In fact, in Global Science mode, the MISR nadir camera is the only one with all four bands at high resolution (275 meters). The other 8 cameras produce red band data at 275 m resolution, but the remaining channels are averaged to 1.1km resolution. MISR also has a Local Science Mode in which all 36 channels are temporarily acquired at 275 m resolution over a selected scene.

Figure 4-1 Artist's Rendition of MISR aboard Terra and sample MISR images.

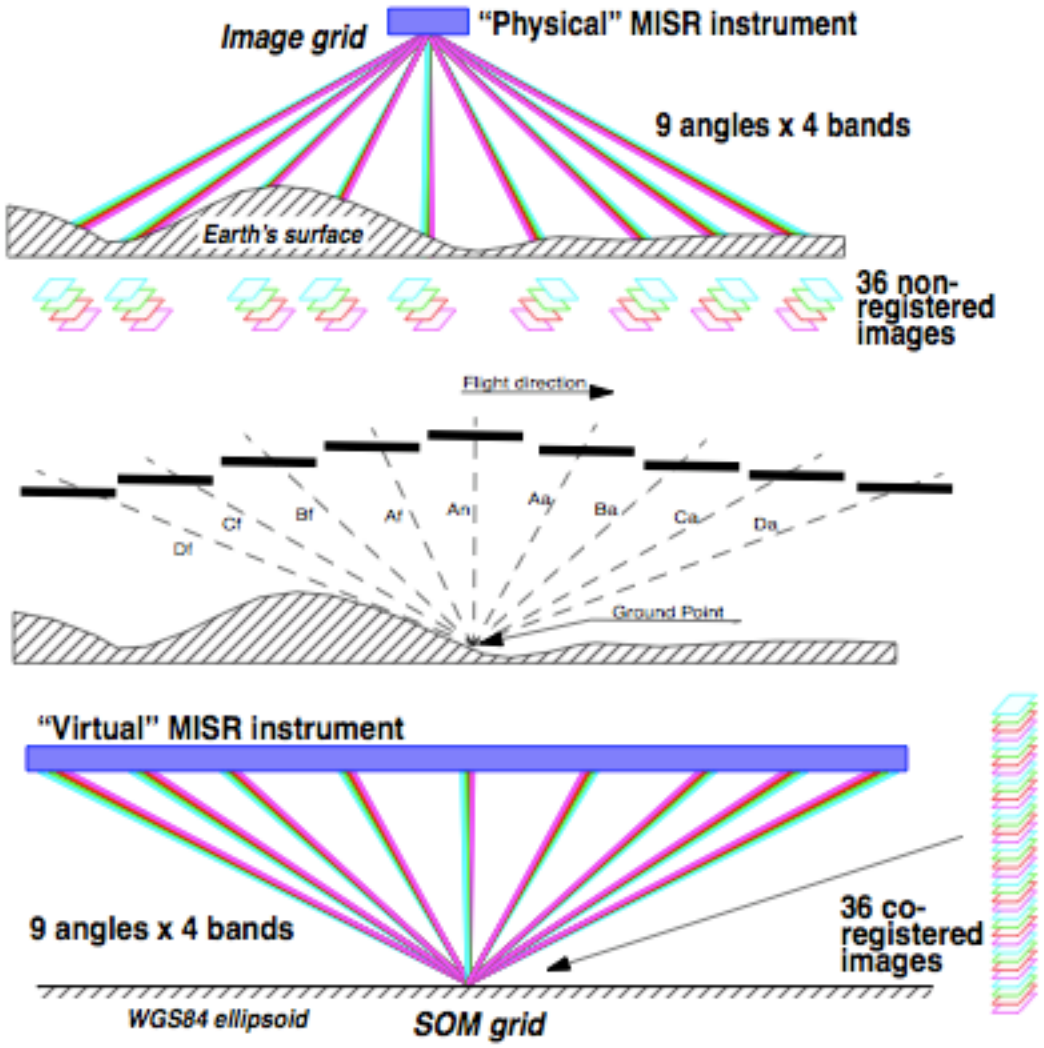


The leftmost panel of Figure [4-1] shows an artist's rendition of the MISR instrument on-board the Terra platform, acquiring data from its nine cameras simultaneously for nine different locations in the four spectral bands. The four panels on the right exhibit the true-color (red, green, blue) images acquired by the nadir and forward 45.6-degree, 60.0-degree, and 70.5-degree cameras, respectively. The sensitivity of the slanted cameras to aerosols (in particular haze) is clearly demonstrated in this sequence.

The diagrams on the next page illustrate the concept that each MISR camera eventually views one ground point at a slightly different time from a different angle as the spacecraft passes over that point. One could imagine an extremely long virtual instrument that could view a scene from many different vantages at once. In essence, this is what MISR does. There is a time lag of

several minutes between the most forward and most aftward observations. This lag is short enough so that scene changes are small, except for wind-driven cloud motion. The problem of co-registration of data from the different cameras is handled by resampling data from each channel onto a common map projection. This common map projection, called SOM, is described in the next section.

Figure 4-2 Diagrams depicting the Multi-Angle concept



4.3 Background: The SOM Projection

MISR acquires data continuously down the entire daylit side of its orbit. The resulting image is a long, narrow "shoestring" swath that covers a vast geographic range. The SOM map projection was designed for Landsat to support continuous images of this extent. SOM is an acronym for

Space Oblique Mercator. In SOM, shape distortion and scale errors are negligible throughout the length of the MISR swath near the satellite ground track. By putting MISR products in the SOM map projection, the complications of projection distortion were removed from geophysical algorithm development and data processing strategy.

Figure 4-3 Sample Partial MISR Swath in SOM vs. Distorted Geographic Lat/Lon

For the majority of MISR data products, SOM is used as the reference map projection. The down side to this scheme is that most users are not familiar with SOM. The following is a brief explanation of the qualitative differences between SOM coordinates and Lat/Lon coordinates. It is by no means a comprehensive introduction to the projection, which is best left to the projection's designer Snyder.[2]

Paths: Polar-orbiting satellites such as Landsat and Terra follow a pattern of orbits which repeats after 233 unique orbits in order to cover the entire globe. Each of the 233 possible orbits is called

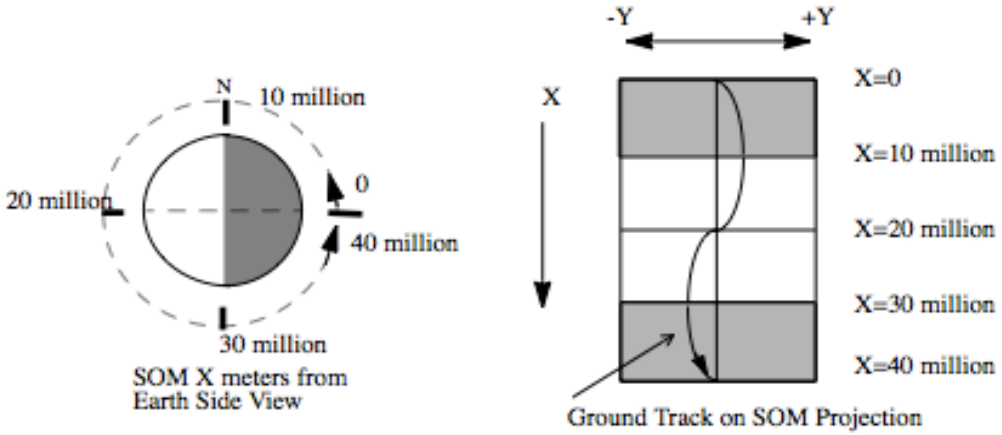
a path. SOM defines a separate projection for each of these 233 paths. For MISR, a path begins at a particular longitude as the satellite crosses the ascending nightside equator. This is referred to as the longitude of the ascending node. Whenever you are dealing with SOM coordinates, you must specify which path is of concern. This path implies a specific longitude of ascending node, which implies a specific SOM projection applicable to that path. All of this information is contained in the MISR product.

(X,Y): SOM coordinates, are called X and Y with units of meters. The X axis points more or less in the direction of satellite groundtrack motion, with the Y axis perpendicular to it. See figure below. The origin of the X axis is at the ascending node equator crossing. To give some feel for the scope of SOM coordinates, X=0 at the dark side equator, X=10 million meters near the North Pole, X=20 million meters at the descending dayside equator crossing, X=30 million meters near the South Pole, and X approaches 40 million meters near the next ascending node. Once the ascending node is reached, X values start over at 0 for the next path/projection.

(Y): SOM coordinate Y values may be negative or positive depending upon the side of the X axis on which the location falls. Do not confuse SOM coordinate Y values with internal MISR sample values, which are always positive. Relevant SOM Y values do not continually increase for the duration of the orbit as X values do. The ground track of the satellite actually follows a nearly sinusoidal curve along the X axis (Y=0); so, at some points on the earth, relevant Y values will be consistently larger than at others, and during a given portion of the orbit, they will be either dominantly positive or negative. The amplitude of the sine curve is about 1 million meters. It is good to note where your area of interest falls on the sine curve in order to determine whether or not you are getting appropriate X,Y values. In general, SOM Y values should be within the range +/- 1 or 2 million meters to stay within the MISR swath at the extremities of the sine curve.

Another SOM attribute to be wary of is that SOM Y is not analogous to longitude. Y indicates how far left or right you are from the SOM X axis, but near the poles this can be a north/south shift, whereas at the equator, changing Y is more east/west in orientation. At the equator, the angle between the SOM X axis and the equator is about 8 degrees.

Figure 4-4 Attributes of SOM coordinates



Width: It is also important to note that if you are "relatively near" the ground track, any lat/lon <-> SOM coordinate conversion you perform will be "reversible" to a reasonable degree of accuracy. On the other hand, if you are very far from the ground track, conversions may not appear to be accurate when reversed. This is caused by a combination of the numerical limitations within the conversion software. (The terms "relatively near" and "very far" are not strictly defined. Suffice it to say that the MISR swath, which extends a few hundred kilometers on either side of the ground track, has been deemed safe by MISR photogrammetrists.) The general lesson to be learned is that you need to know which path you are on, and where the orbit ground track is, and then stay near it in order to get satisfactory performance with SOM coordinate conversions.

Line/Sample: Pixels in a MISR product are arranged in a regular 2-D array in SOM space. The indices to the array are called absolute line and sample, where line increases from top to bottom and sample increases from left to right. Therefore, if you know the SOM X,Y coordinates of any one point in the swath, you can deduce the SOM coordinates of any other point in the swath if you know the pixel resolution and absolute line/sample offset. Beware that the line and sample values in a MISR file are block-relative. They are not absolute.

Blocks: There is one added complication to SOM in MISR products. In order to simplify the job of processing and storing data over this immense geographical area, each MISR path was cut up into a series of pre-defined, uniformly-sized SOM boxes along the ground track. Each box-shaped region is called a Block. MISR blocks are similar to Landsat rows. Block-relative line and sample restart at 0,0 at the top left corner of each block. Therefore, a trivial conversion is required to determine the SOM coordinates of a given pixel in a MISR file, specified as (block, line, sample). Once SOM (X,Y) meters are known, GCTP coordinate conversion software[3] may be used convert between SOM coordinates and Lat/Lon.

4.4 Background: HDF-EOS

HDF-EOS Grid: All MISR products are in the HDF 4 format. However, it is easier to interpret the structures in MISR files if one realizes that they are actually HDF-EOS structures. The EOS project designed specialized data types and access routines on top of HDF 4. These datatypes and the software libraries that read and write them are referred to collectively as HDF-EOS. The earliest products in the MISR production chain (L1A and L1B1) are single-camera HDF-EOS Swath data types. In HDF terminology, a Swath is just a big long SDS or array. Swaths contain no geolocation information other than the time at which a line was acquired. Most users are not interested in this raw Swath format. Instead, they wish to access map-projected data so that they can compare different measurements at a particular geographic location. The HDF-EOS data type for map-projected data is called "Grid." The HDF-EOS Grid model include metadata structures to store and software support to manipulate orbital parameters and map projection parameters along with the data. Most MISR data products consist of HDF-EOS Grid structures along with specific EOS-defined metadata entities. In theory, it should be very easy to retrieve location information for any MISR pixel in any map projection desired.

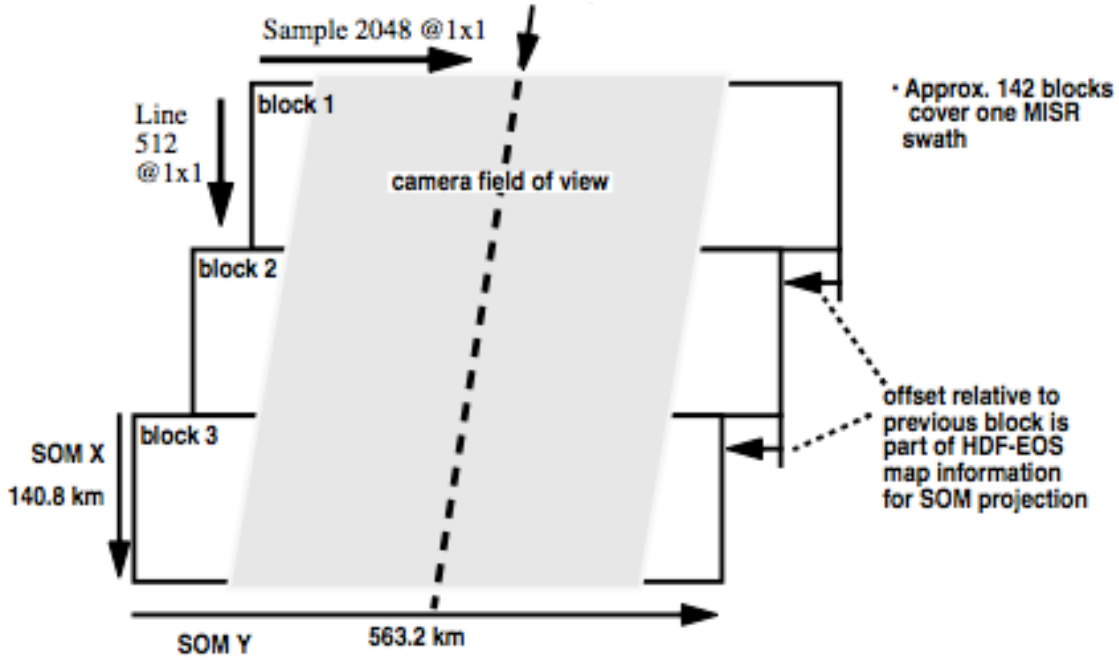
"Stacked Block" Grid: MISR data doesn't fit into the HDF-EOS Grid model very well. The designers of HDF-EOS Grid envisioned small, rectangular maps. Each MISR data swath is a long, curving shoestring which cuts through a huge range of latitudes and longitudes. A single Grid (rectangle) encompassing a typical MISR orbit would have to span the entire globe to bound the shoe string. Grids this large were thought to be intractable in the early days of EOS. So, the EOS project defined a special extension to the Grid model for MISR called "stacked block." A MISR block is an arbitrarily-sized SOM rectangle on the Earth. A MISR data swath could be contained in a series of adjacent blocks instead of one huge Grid. The blocks are stacked one on top of another. The lateral offset from block to block is not constant. (See the diagram in Figure 4-5 below)

Block Offsets: A block may be placed directly beneath the one above it, or it may be shifted by an integral multiple of 17.6 km in the lateral (+/- SOM Y) direction. These shifts are all pre-defined so that the blocks comfortably span the Terra ground track to encompass a MISR data swath. Each block can be thought of as a separate HDF-EOS Grid, with the entire series comprises a single Grid structure in an HDF file. A MISR HDF-EOS Grid is therefore defined for a given path by the coordinates of the first (top) MISR block along with the standard projection metadata and a special array of offsets defining the locations of all subsequent blocks. Each subsequent block may be treated by the user as an independent Grid, provided that its position relative to the top block can be calculated. The HDF-EOS stacked block model involves automated storage and retrieval of offset metadata, but it does not perform offset calculations. The user must perform these calculations and assemble the desired set of blocks accordingly to obtain a mutli-block map.

Use Patterns: Two primary access paradigms have been noticed. Some users pull one block at a time out of a MISR product and use it in SOM space. Such users are often MISR-centric and manipulate data with their own software. Other users wish to pull out a geographically-defined subset of MISR data which may be smaller than one block or which may entail several blocks. The later type of user often prefers to work with the data in some map projection other than SOM

with a commercial Image Processing or GIS software tool. In either case, familiarity with the MISR internal data representation is helpful.

Figure 4-5 Depiction of MISR Stacked Block Grid



Surprises: There are several idiosyncrasies worth noting at this point. Notice that the MISR image does not fill the entire block. The unused edges of the block contain fill value. The extra room is needed for several reasons. First, the spacecraft ground track is actually inclined slightly with respect to the SOM X axis. Second, not all nine camera footprints overlap precisely. Third, the union of camera footprints gets wider within the blocks at higher latitudes as the overlap gets smaller. This is due to Earth rotation and fixed camera geometry. Finally, each block may be shifted left or right of those adjacent to it by some multiple of 17.6 km in order to follow the groundtrack.

180 Blocks: For each MISR orbit path, a set of 180 SOM blocks has been predefined. The 180 blocks cover a range larger than the daylit Earth in order to account for seasonal variations in the positions of day/night terminators. MISR only acquires data on the dayside of the terminator. For a single orbit, the terminator-to-terminator range in blocks is roughly 142. Therefore, MISR files contain data structures representing 180 blocks, but data is only found in 142 or fewer of these blocks. During winter months, for instance, the first 20 blocks of a MISR Grid may be vacant.

Figure 4-6 MISR Blocks Follow the Curving Swath

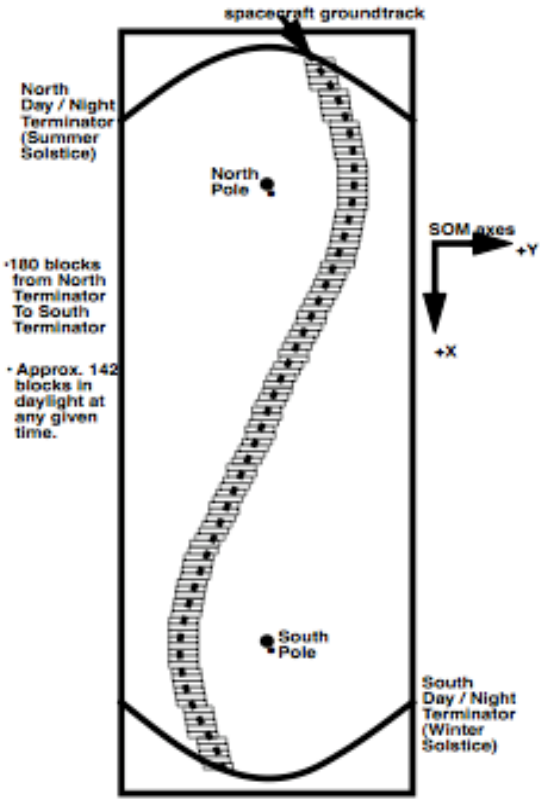


Diagram representing the 180 block series covering a MISR path.

Actual full-length MISR swath image in SOM space for perspective.

4.5 Coordinate Conversions

The following three subsections [4.5.1], [4.5.2] and [4.5.3] describe the steps required to perform accurate coordinate conversions on pixel locations in a MISR file. The descriptions include the extraction of metadata necessary to perform the conversions. A complete description of MISR file metadata relevant to these conversions is provided in section [4.6]. Example function calls relevant to these conversions in the C language are printed in section [4.7]. An assumption is made that the user has access to the HDF-EOS libraries, including the GCTP map projection library[3]

Forward Conversion: Lat/Lon->(block, line, sample) = [Lat/Lon-> SOM ->(block, line, sample)]

Given a position specified by Lat/Lon, two steps are required to determine the MISR block, line and sample coordinates. 1) Convert Lat/Lon to SOM(X,Y) meters using the GCTP conversion software. 2) Then convert SOM(X,Y) to MISR (block, line, sample). Forward conversion is useful for resam pling MISR data to another map projection. GCTP supports conversions from Lat/Lon to other projections using the same metadata required for the SOM->Lat/Lon conversion. Beware that the line and sample results for an arbitrary position may be fractional.

Inverse Conversion: (block, line, sample)->Lat/Lon = [(block, line, sample)->SOM->Lat/Lon] Given a MISR pixel, as specified by block number, line number within block and sample number within block, two inverse steps are required to determine the Latitude and Longitude of the pixel center. 1) Convert (block, line, sample) to SOM(X,Y) meters. 2) Then convert SOM(X,Y) to Lat/ Lon using the GCTP conversion software. The orbit path must be known a priori; metadata from the MISR product file in question is required; and the HDF-EOS software library along with GCTP must be utilized. Inverse conversion is useful for determining the location of features within native MISR data for comparison with other datasets.

Precision: Coordinate Conversions are reversible (Forward<->Inverse) with reasonable numerical precision for positions near the satellite groundtrack. Positions within the MISR data swath are thus reversible. Positions that are not near the satellite groundtrack are often better-described on another path and SOM projection. Coordinate conversions involving such positions are often not reversible. The causes of this limitation include map projection distortion as well as numerical approximation used in the algorithms in GCTP. The example function calls in section [4.7] provide a good use case to emulate.

4.5.1 SOM <-> Lat/Lon

Inverse: SOM -> Lat/Lon Given a pixel's position in SOM (X,Y) meters, assuming a particular orbit path, the following steps should be used to determine the corresponding Latitude and

Longitude.

a) Choose a MISR product file with the appropriate orbit path and read from it the HDF-EOS projection params using the call GDprojinfo(). The projection parameters define the SOM projection used for this path. Users wishing to avoid HDF-EOS may refer to the Metadata Details section for choices for direct HDF reads of projection parameter info.

b) Convert the SOM coordinates to Lat/Lon with the GCTP library calls inv_init() and sominv(). inv_init takes the projection parameters from step a. as arguments.[1]
Beware that the order of the arguments to sominv(som_x, som_y, &lon, &lat) is neither intuitive nor well-documented.[2] See section [4.7] for examples.

Forward: Lat/Lon -> SOM Given a pixel's position in Lat/Lon coordinates assuming a particular MISR orbit path, the following steps should be used to determine the corresponding coordinates in SOM (X,Y) meters.

a) Choose a MISR product file with the appropriate orbit path and read from it the HDF-EOS projection params using the call GDprojinfo(). The projection params define the SOM projection used for this path. Users wishing to avoid HDF-EOS may refer to the Metadata Details section for choices for direct HDF reads of projection parameter info.

b) Convert the Lat/Lon coordinates to SOM with the GCTP library calls for_init() and somfor(). for_init takes the projection parameters from step a. as arguments.[3]
Beware that the order of the arguments to somfor(lon, lat, &som_x, &som_y) is neither intuitive nor well-documented.

4.5.2 Inverse: MISR(block, line, sample) -> SOM(X,Y)

Given a MISR pixel specified by (block, line, sample), assuming a particular orbit path, the following steps should be used to determine the corresponding SOM(X,Y) coordinates in meters.

a) Pick a MISR file to read which corresponds to the orbit path in question.

b) Read origin block coords and block/pixel sizes for a band using HDF-EOS GDgridinfo()[4][5]

[1] The GCTP coordinate conversion library provides routines for converting between lat/lon and many other map projections.MISR uses the GCTP SOM projection A by default based on the projection parameter values. SOM A specifies the inclination angle and longitude of the ascending node unlike SOM B, which uses path number and is specific to Landsat.
[2] In HDF-EOS, Lat/Lon coordinates are specified in degrees. GCTP functions expect Lat/Lon in radians.
[3] GDprojinfo() actually returns the HDF-EOS projection parameters array which corresponds directly to the one required for the GCTP SOM initializations; but in addition, it returns the projection code (22 for SOM), the zone code (unused for SOM) and the spheroid code (WGS84 ellipsoid 12), all of which are required to call the GCTP routines. See [4] for more info.
[4] Though it is recommended, the user does not have to use HDF-EOS routines to read relevant metadata values. If

From the origin coords (ulc[], lrc[]) and the sizes (Xdim, Ydim), compute the following values:

ULC.x = ulc[0] !! JUST DO THIS. It is not a typo.
ULC.y = lrc[1] !! Swapping ULC.y and LRC.y is a
LRC.y = ulc[1] !! side-effect of an unusual definition of
LRC.x = lrc[0] !! lrc vs. ulc in HDFEOS.
Sx = (LRC.x - ULC.x) / Xdim !! Size of pixel in line direction in meters
Sy = (LRC.y - ULC.y) / Ydim !! Size of pixel in sample direction in meters

c) Adjust ULC coordinates from pixel corner to pixel center.
ULC.xc = ULC.x + (Sx / 2.0)
ULC.yc = ULC.y + (Sy / 2.0)

d) Read the block offsetArray using the HDF-EOS call GDblkSOMoffset(). It returns an array of offsets specified in pixels at Sx resolution. Each offset is relative to the block above. The HDF-EOS call actually reads from the vdata structure _BLKSOM:<gridname>.

e) Calculate SOM.x and SOM.y for BlockNumber(**b**), pixel(**line, sample**) as:
SOM.x = ULC.xc + [(b - 1) * Xdim * Sx] + (line * Sx)

SOM.y = ULC.yc + [sample + offset] * Sy where **offset** = $\sum_{i=0}^{b-2} offsetArray[i]$

4.5.3 Forward: SOM(X,Y) -> MISR (block, line, sample)

Given a position in SOM(X,Y) meters, assuming a particular orbit path, the following steps should be used to determine the corresponding MISR pixel in (block, line, sample) coordinates.

a) Pick a MISR file to read which corresponds to the orbit path in question.

b) Read origin block coords and block/pixel sizes for a band using HDF-EOS GDgridinfo().

From the origin coords (ulc[], lrc[]) and the sizes (Xdim, Ydim), compute the following values:

---

some other means of accessing MISR files is desired, see "Metadata Details" on page 358. Beware that the definitions of ULC.y and LRC.y are not the same for all metadata fields!

[5] Each MISR band may be of a different resolution. So, you need to obtain the 1st block origin coordinates and the block and pixel size information using the HDF-EOS call GDgridinfo() which reads from the textual StructMetaData. StructMetaData is present in all MISR HDF files.The GDgridinfo() call returns:

ULC.x = ulc[0] !! JUST DO THIS. It is not a typo.
ULC.y = lrc[1] !! Swapping ULC.y and LRC.y is a
LRC.y = ulc[1] !! side-effect of an unusual definition of
LRC.x = lrc[0] !! lrc vs. ulc in HDFEOS.
Sx = (LRC.x - ULC.x) / Xdim !! Size of pixel in line direction in meters
Sy = (LRC.y - ULC.y) / Ydim !! Size of pixel in sample direction in meters

c) Adjust ULC coordinates from pixel corner to pixel center.
ULC.xc = ULC.x + (Sx / 2.0)
ULC.yc = ULC.y + (Sy / 2.0)

d) Convert SOM meters to local SOM representation in pixels relative to first block in this file.
SOM.xpix = (SOM.x - ULC.xc) / Sx
SOM.ypix = (SOM.y - ULC.yc) / Sy

e) Determine the block number.
Block = floor((SOM.xpix + 0.5) / Xdim) + 1 !! Coords are within this block[6].

f) Determine line number within predetermined block.
Line = SOM.xpix - ((Block - 1) * Xdim)

g) Read the block offsetArray using the HDF-EOS call GDblkSOMoffset(). It returns an array of offsets specified in pixels at Sx resolution. Each offset is relative to the block above.

h) Determine the sample coordinate within the predetermined block.
Sample = SOM.ypix - offset where **offset** = $\sum_{i=0}^{Block-2} offsetArray[i]$

4.6 Metadata Details

There are a handful of characteristics of MISR Grid files which are critical for accurate and efficient data access. For instance, each MISR Grid is 180 blocks in length; however, roughly 142 sequential blocks actually contain valid data. File-Global metadata attributes Start_block and End Block tell the user the range and location of valid blocks within the 180 defined in the hdf file. The externally mandated definitions of HDF-EOS Grid ULC and LRC are inverted for SOM! Therefore some coordinates in the table must be swapped. They are noted. Other custom metadata fields are very handy to know about. The following table allows users without access to HDF-EOS libraries to obtain critical geo-related metadata from the most convenient sources.

[6] Real code should check Block, Line and Sample coordinates against valid ranges. Also, do not be fooled by X/Y inversion from standard video definition. See section "Background: The SOM Projection".

Note: All metadata coordinates refer to outside corner locations, not to centers of corner-pixels. Metadata values base-1 by convention. Block number is the most common example. Software developers often expect base-0, so beware

Table 4-1 MISR Metadata Sources

Metadata Location	Structural	Core	Gbl. File Attr.*	Grid Attr.*	Per Block Common Vdata*
Projection Parameters	ProjParams		various SOM_...		
OriginBlock Coordinates	UperLeftPointMtrs LowerRightMtrs - SOM meters - Beware! Y coor dinates swapped				Block_coor_ulc_som: _meter.x _meter.y Block_coor_lrc_som: _meter.x _meter.y
All Block Coordinates		GRINGPOINTLONGITUDE GRINGPOINTLATITUDE - Lat/Lon - Order(ULC, LLC, LRC, URC)			see above
Pixels per Block	XDim YDim			Block_size.size_x Block_size.size_y	
Pixel Size	(Derivable)			Block_size.resolution _x Block_size.resolution _y	
Block Offset Ary.	Special HDF-EOS vdatas, one per grid named _BLKSOM:<gridname>				
Valid Block Range			Start_block End block		
*** indicates MISR-unique metadata structure vs. standard ECS-required metadata.**					

4.7 Example Function Calls

The examples in this section illustrate the coordinate conversion scenarios described in this appendix. The GCTP software package is used to convert between SOM and Lat/Lon coordinates. C Programmers often understand C better than English. These examples are intended to address detailed questions about function interfaces, units and adjustments. Such issues are more easily addressed in this way than in prose. These examples are strictly intended for educational purposes.

In the following example scenario, map projection parameters are read from a real MISR HDF-

EOS Grid file. Example coordinate conversions are made, and the function calls to HDF and GCTP library routines are shown in the proper sequence with proper arguments.

misrcoordex.c outlines a series calls to the example functions, HDF-EOS and the GCTP library to perform full MISR->Lat/Lon and Lat/Lon->MISR conversions.

misr_init() illustrates setup steps which simplify the remaining examples. It should be called done prior to calling misrfor() or misrinv(). These steps, mentioned in sections [4.5.2] and [4.5.3], include converting from relative to absolute coordinates and the swapping of ulc/rlc values.

misrfor() illustrates the forward conversion SOM(X,Y) -> (block, line, sample) described in Section [4.5.3].

misrinv() illustrates the inverse conversion (block, line, sample) -> SOM(X,Y) described in Section [4.5.2].

The files misrproj.h and errormacros.h are a headers included by the other examples to remove extraneous code so that the examples are more concise.

The Makefile shows how to include from and link to the necessary pieces of the GCTP, HDF and HDF-EOS libraries needed to complete the scenario.

The basic scenario outlined in misrcoordex.c is:

a. Use HDF-EOS to read necessary info from a MISR product.

b. Call misr_init() and som_init() once with the info read from the MISR product.

c. use combinations of (somfor()+misrfor()) or (misrinv()+sominv()) to perform as many for ward or inverse coordinate conversions as desired on this orbit path.

4.7.1 misrcoordex.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hdf.h>
#include <HdfEosDef.h>
#include <proj.h>
#include “misrproj.h”
#include “errormacros.h”

#define MAXNDIM          10

typedef struct {
```

```
  int  block;
  float line;
  float sample;
} pts_t;

int npts = 40;
pts_t pts[] = { {   1,      -0.5,      -0.5 }, \
                {   1, -0.499999, -0.499999 }, \
                {   1,       0.0,       0.0 }, \
                {   1,       0.5,       0.5 }, \
                {   1,     127.0,     511.0 }, \
                {   1,     127.5,     511.5 }, \
                {   1,     511.0,    2047.0 }, \
                {   1,     511.5,    2047.5 }, \
                {   1,    101.97,     64.23 }, \
                {   1,     101.0,      64.0 }, \
            {  65,      -0.5,      -0.5 }, \
                {  65, -0.499999, -0.499999 }, \
                {  65,       0.0,       0.0 }, \
                {  65,       0.5,       0.5 }, \
                {  65,     127.0,     511.0 }, \
                {  65,     127.5,     511.5 }, \
                {  65,     511.0,    2047.0 }, \
                {  65,     511.5,    2047.5 }, \
                {  65,    101.97,     64.23 }, \
                {  65,     101.0,      64.0 }, \
            {  91,      -0.5,      -0.5 }, \
                {  91, -0.499999, -0.499999 }, \
                {  91,       0.0,       0.0 }, \
                {  91,       0.5,       0.5 }, \
                {  91,     127.0,     511.0 }, \
                {  91,     127.5,     511.5 }, \
                {  91,     511.0,    2047.0 }, \
                {  91,     511.5,    2047.5 }, \
                {  91,    101.97,     64.23 }, \
                {  91,     101.0,      64.0 }, \
            { 180,      -0.5,      -0.5 }, \
                { 180, -0.499999, -0.499999 }, \
                { 180,       0.0,       0.0 }, \
                { 180,       0.5,       0.5 }, \
                { 180,     127.0,     511.0 }, \
                { 180,     127.5,     511.5 }, \
                { 180,     511.0,    2047.0 }, \
                { 180,     511.5,    2047.5 }, \
                { 180,    101.97,     64.23 }, \
                { 180,     101.0,      64.0 }, \
};

int main(int argc, char *argv[]) {

  int32                 fid = FAIL;
  int32                 gid = FAIL;
  int                   igrid, i;
  int32                 ngrid;
  int32                 nline, nsample;
  double        lat_r, lon_r;
  double        savelon_r1, savelon_r2;
  double        somx, somy;
  int                   b;
  float                 l, s;
  int32                 strbufsize;
  char                  *filepath;
```

```
char            **gridname;
char            *gridlist;
float64     ulc[2], lrc[2];
int32           spherecode, zonecode, projcode;
float64     projparam[NPROJ];
float32     offset[NOFFSET];
long            iflg;
int             status;
char            difflg;
int32           dim[MAXNDIM];
char            dimlist[STRLEN];
intn            hdfeos_status_code;
void            *mem_status_code;
long            (*for_trans[MAXPROJ+1])();
long            (*inv_trans[MAXPROJ+1])();

/* --------------- */
/* Check arguments */
/* --------------- */

if (argc != 2) {
  fprintf(stderr, "Usage: %s hdfeos_grid_file\n", argv[0]);
  exit(1);
}
filepath = argv[1];

/* ------------------------------------------------------ */
/* Inquire and allocate memory for the hdfeos gridnames */
/* This is only require if you need the gridnames       */
/* ------------------------------------------------------ */

hdfeos_status_code = GDinqgrid(filepath, NULL, &strbufsize);
HDFEOS_ERROR_CHECK("GDinqgrid");

mem_status_code = gridlist = (char *)malloc(strbufsize+1);
MEM_ERROR_CHECK("malloc");

hdfeos_status_code = ngrid = GDinqgrid(filepath, gridlist, NULL);
HDFEOS_ERROR_CHECK("GDinqgrid");

mem_status_code = gridname = (char **)malloc(ngrid * sizeof(char *));
MEM_ERROR_CHECK("malloc");

gridname[0] = strtok(gridlist, ",");
for (igrid = 1; igrid < ngrid; igrid++) gridname[igrid] = strtok(NULL,",");

/* ------------------------ */
/* Open the hdfeos grid file */
/* ------------------------ */

hdfeos_status_code = fid = GDopen(filepath, DFACC_READ);
HDFEOS_ERROR_CHECK("GDopen");

/* ---------------------------------------- */
/* Loop through all the grids because I can */
/* ---------------------------------------- */

for (igrid = 0; igrid < ngrid; igrid++) {

  /* --------------------------- */
  /* Attach to the grid of choice */
  /* --------------------------- */
```

```
hdfeos_status_code = gid = GDattach(fid, gridname[igrid]);
HDFEOS_ERROR_CHECK("GDattach");

/* ---------------------------------------------------------------- */
/* Inquire grid dimensions to check number of blocks                */
/* Inquire grid info to get the number of lines/sample and ulc/lrc */
/* Inquire SOM relative block offsets                               */
/* Initialize misr block/line/sample projection routines            */
/* ---------------------------------------------------------------- */

hdfeos_status_code = GDinqdims(gid, dimlist, dim);
HDFEOS_ERROR_CHECK("GDinqdims");
if (dim[0] != NBLOCK) ERROR("File does not have 180 blocks");

hdfeos_status_code = GDgridinfo(gid, &nline, &nsample, ulc, lrc);
HDFEOS_ERROR_CHECK("GDgridinfo");

hdfeos_status_code = GDblkSOMoffset(gid, offset, NOFFSET, "r");
HDFEOS_ERROR_CHECK("GDblkSOMoffset");

status = misr_init(NBLOCK, nline, nsample, offset, ulc, lrc);
if (status) ERROR("misr_init");

printf("\nFilename (path/orbit): %s\n", filepath);
printf("Gridname: %s\n", gridname[igrid]);
printf("Lines/Samples: (%d, %d)\n", nline, nsample);
printf("ULC (x,y) (m): (%f, %f)\n", ulc[0], ulc[1]);
printf("LRC (x,y) (m): (%f, %f)\n", lrc[0], lrc[1]);
printf("Block offsets: (%f", offset[0]);
for (i = 1; i < NOFFSET; i++) printf(", %f", offset[i]);
printf(")\n");

/* ---------------------------------------------------------------- */
/* Inquire grid projection info to get project codes/parameters */
/* Initialize gctp SOM forward and inverse projection routines  */
/* ---------------------------------------------------------------- */

hdfeos_status_code = GDprojinfo(gid, &projcode, &zonecode,
                                &spherecode, projparam);
HDFEOS_ERROR_CHECK("GDprojinfo");

for_init((long)projcode, (long)zonecode, (double*)projparam,
         (long)spherecode, NULL, NULL, &iflg, for_trans);
if (iflg) ERROR("for_init");

inv_init((long)projcode, (long)zonecode, (double*)projparam,
         (long)spherecode, NULL, NULL, &iflg, inv_trans);
if (iflg) ERROR("inv_init");

printf("GCTP projection code: %d\n", projcode);
printf("GCTP zone code (not used for SOM): %d\n", zonecode);
printf("GCTP sphere code: %d\n", spherecode);
printf("GCTP projection parameters: (%f",projparam[0]);
for (i = 1; i < NPROJ; i++) printf(", %f", projparam[i]);
printf(")\n");

/* ---------------------------------------------------------------- */
/* Detach from the grid because we don't need it anymore in this example */
/* We would need it if we go on to access fields, so don't detach here   */
/* ---------------------------------------------------------------- */
```

```
if (gid != FAIL) GDdetach(gid);

/* ---------------------------------------- */
/* Loop over some inverse transformations   */
/* (b,l.l,s.s) -> (X,Y) -> (lat,lon)        */
/* and over some forward transformations    */
/* (lat,lon) -> (X,Y) -> (b,l.l,s.s)        */
/* ---------------------------------------- */

printf("   (blk,    line ,    sample )   "
     "(       SOM X    ,       SOM Y    )   "
          "(    Lat   ,     Lon   )\n");

for (i = 0; i < npts; i++) {

  b = pts[i].block;
  l = pts[i].line;
  s = pts[i].sample;

  /* ---------------------------------------------------------- */
  /* Inverse transformation (b,l.l,s.s) -> (X,Y) -> (lat,lon) */
  /* ---------------------------------------------------------- */

  misrinv(b, l, s, &somx, &somy);      /* (b,l.l,s.s) -> (X,Y) */
  sominv(somx, somy, &lon_r, &lat_r);  /* (X,Y) -> (lat,lon) */

  printf("%2d: (%3d,%11.6f,%12.6f) -> (%17.6f,%17.6f) -> "
       "(%10.6f,%11.6f) --|\n",
            i, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

  /* ---------------------------------------------------------- */
  /* Forward transformation (lat,lon) -> (X,Y) -> (b,l.l,s.s) */
  /* ---------------------------------------------------------- */

  somfor(lon_r, lat_r, &somx, &somy);  /* (lat,lon) -> (X,Y) */
  misrfor(somx, somy, &b, &l, &s);     /* (X,Y) -> (b,l.l,s.s) */

  if (b != pts[i].block) diffflg = '*';
  else diffflg = ' ';

  printf("  %c (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) <- "
            "(%10.6f,%11.6f) <-|\n",
            diffflg, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

  /* ---------------------------------------------------- */
  /* Save the longitude of block 91 to find location of */
  /* equator crossing                                   */
  /* ---------------------------------------------------- */

  if (pts[i].block == 91 &&
          pts[i].line == 0.0 &&
          pts[i].sample == 0.0) {
        savelon_r1 = lon_r;
  }
  if (pts[i].block == 91 &&
          pts[i].line == (float)(nline-1) &&
          pts[i].sample == (float)(nsample-1)) {
        savelon_r2 = lon_r;
  }
}

/* ---------------------------------------------------- */
```

```
/* Determine block/line/sample of the equator crossing */
/* approximately in the center of the block            */
/* --------------------------------------------------- */

lat_r = 0.0;
if (savelon_r1 < 0.0 && savelon_r2 > 0.0 ||
          savelon_r1 > 0.0 && savelon_r2 < 0.0) {
  lon_r = (savelon_r1 - savelon_r2) / 2.0;
} else {
  lon_r = (savelon_r1 + savelon_r2) / 2.0;
}

/* ------------------------------------------------------------ */
/* Forward transformation (lat,lon) -> (X,Y) -> (b,l.l,s.s) */
/* ------------------------------------------------------------ */

somfor(lon_r, lat_r, &somx, &somy);  /* (lat,lon) -> (X,Y) */
misrfor(somx, somy, &b, &l, &s);     /* (X,Y) -> (b,l.l,s.s) */

printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) <- "
            "(%10.6f,%11.6f) = equator crossing\n",
            npts, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

/* -------------------------------------- */
/* Extreme upper left corner (not center) */
/* -------------------------------------- */

somx = ulc[0];
somy = lrc[1];                    /* Notice the switch from ulc[1]. */

sominv(somx, somy, &lon_r, &lat_r);
misrfor(somx, somy, &b, &l, &s);

printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) -> "
            "(%10.6f,%11.6f) = ulc of block 1\n",
            npts+1, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

/* ------------------- ------------------- */
/* Extreme lower right corner (not center) */
/* ------------------- ------------------- */

somx = lrc[0];
somy = ulc[1];                    /* Notice the switch from lrc[1]. */

sominv(somx, somy, &lon_r, &lat_r);
misrfor(somx, somy, &b, &l, &s);

printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) -> "
            "(%10.6f,%11.6f) = lrc of block 1\n",
            npts+2, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

/* -------------------------------------- */
/* Origin of SOM projection for this path */
/* -------------------------------------- */

somx = 0.0;
somy = 0.0;

sominv(somx, somy, &lon_r, &lat_r);
misrfor(somx, somy, &b, &l, &s);

printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) -> "
```

```
            "(%10.6f,%11.6f) = SOM origin (long of asc node)\n",
            npts+3, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

  /* ------------------------------------------------- */
  /* Origin of SOM projection plus 180 degrees longitude */
  /* ------------------------------------------------- */

  lat_r = 0.0;
  lon_r = (lon_r > 0.0 ? lon_r - (180.0*D2R) : lon_r + (180.0*D2R));

  somfor(lon_r, lat_r, &somx, &somy);
  misrfor(somx, somy, &b, &l, &s);

  printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) <- "
            "(%10.6f,%11.6f) = SOM origin plus 180 in long.\n",
            npts+4, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

  /* ---------------------------------------- */
  /* Equator crossing using SOM X from above */
  /* ---------------------------------------- */

  somy = 0.0;

  sominv(somx, somy, &lon_r, &lat_r);
  misrfor(somx, somy, &b, &l, &s);

  printf("%2d: (%3d,%11.6f,%12.6f) <- (%17.6f,%17.6f) -> "
            "(%10.6f,%11.6f) = equator crossing\n",
            npts+5, b, l, s, somx, somy, lat_r * R2D, lon_r * R2D);

 }

 if (fid != FAIL) GDclose(fid);
 if (gridlist) free(gridlist);
 if (gridname) free(gridname);

 printf("\nNotes:\n\n"
"1) Given a block, fractional line and fraction sample triplet the\n"
"   following transformations performed:\n\n"
"    Inverse transformation:     (b,l.l,s.s) -> (X,Y) -> (lat,lon) -|\n"
"                          |-----------------------------------|\n"
"    Forward transformation: |-> (lat,lon) -> (X,Y) -> (b,l.l,s.s)\n\n"
"2) The transforms marked with a * did not reproduce the same\n"
"   answer either because of rounding errors in the GCTP codes or because\n"
"   they are out of bounds of the particular grid.  The misr transform\n"
"   routines (misr_init, misrfor and misrinv) are designed to handle out\n"
"   of bounds conditions and return all -1's.  This enables a resampling\n"
"   routine to determine whether resampling can be done or not, if these\n"
"   routines are used for reprojection.\n\n"
"3) Notice that the ULC Y/LRC Y values returned by gridinfo are incorrectly \n"
"   switched when compared to transform number 0, 5 or 7 (depending\n"
"   on resolution).\n\n"
"4) Also note that the ULC/LRC values returned by gridinfo are for block 1\n"
"   extreme pixel edges (not pixel centers).\n\n"
"5) Note that SOM X is always increasing as blocks increase (in fact,\n"
"   SOM X is zero meters at the longitude of the ascending node - the\n"
"   5th parameter of projection paramters).  SOM Y tends to be mostly\n"
"   positive in the Northern blocks and negative in the Southern blocks.\n"
"   Each SOM path is a separate projection with the origin at the\n"
"   night side equator and the longitude of the ascending node.\n\n"
"6) The block offsets are the number of 1.1km subregions from the\n"
"   previous block.  The first offset is relative first block.\n\n"
```

```
"7) The 4th and 5th projection parameter are in the format of packed\n"
"   dddmmmsss.ss as documented in the GCTP codes (see paksz.c).\n\n"
"8) MISR uses the GCTP SOM projection A which specifies the inclination\n"
"   angle and longitude of the ascending node instead of path number\n\n"
"9) The last six transformations compute various special case locations.\n"
"   Note the direction of the transform arrows.  Can you determine why the\n"
"   lrc of block 1 is actually in block 2?  Hint: it is not the pixel\n"
"   center, but rather the edge.\n\n"
"10) Last note.  Remember that the SOM projection is singular at the poles\n"
"    and thus undefined there.\n\n"
            );
  exit(0);
}
```

4.7.2 misr_init.c

```
#include "misrproj.h"                    /* Prototype for this function */
#include "errormacros.h"          /* Error macros */
int nb;
int nl;
int ns;
float absOffset[NBLOCK];
float relOffset[NBLOCK-1];
double ulc[2];
double lrc[2];
double sx;
double sy;
double xc;
double yc;

#define FUNC_NAMEm "misr_init"

int misr_init(
const int     nblock,        /* Number of blocks */
const int         nline,         /* Number of lines in a block */
const int         nsample,       /* Number of samples in a block */
const float               relOff[NOFFSET],/* Block offsets */
const double              ulc_coord[],    /* Upper left corner coord. in meters */
const double              lrc_coord[]     /* Lower right corner coord. in meters */
)
{
  int                     i;                      /* Offset index */
  char                    msg[STRLEN];      /* Warning message */

/* Argument checks */

  if (nblock < 1 || nblock > NBLOCK) {
    sprintf(msg,"nblock is out of range (1 < %d < %d)", nblock, NBLOCK);
    WRN_LOG_JUMP(msg);
  }

/* Convert relative offsets to absolute offsets */

  absOffset[0] = 0.0;
  for (i = 1; i < NBLOCK; i++) {
```

```
   absOffset[i] = absOffset[i-1] + relOff[i-1];
   relOffset[i-1] = relOff[i-1];
  }

/* Set ulc and lrc SOM coordinates */
/* Note; ulc y and lrc y are reversed in the structural metadata. */

  ulc[0] = ulc_coord[0];
  ulc[1] = lrc_coord[1];
  lrc[0] = lrc_coord[0];
  lrc[1] = ulc_coord[1];

/* Set number of blocks, lines and samples */

  nb = nblock;
  nl = nline;
  ns = nsample;

/* Compute pixel size in ulc/lrc units (meters) */

  sx = (lrc[0] - ulc[0]) / nl;
  sy = (lrc[1] - ulc[1]) / ns;

/* Adjust ulc to be in the center of the pixel */

  xc = ulc[0] + sx / 2.0;
  yc = ulc[1] + sy / 2.0;

  return(0);

 ERROR_HANDLE:
  return(1);
}
```

4.7.3 misrfor.c

```
#include “misrproj.h”                    /* Prototype for this function */
#include “errormacros.h”        /* Error macros */
#include <math.h>                /* Prototype for floor */

extern int nb;
extern int nl;
extern int ns;
extern float absOffset[NBLOCK];
extern double ulc[2];
extern double lrc[2];
extern double sx;
extern double sy;
extern double xc;
extern double yc;

#define FUNC_NAMEm “misrfor”

int misrfor(
const double            x,                    /* Output SOM X coordinate */
```

```
const double        y,                  /* Output SOM Y coordinate */
int*                block,      /* Input block */
float*              line,       /* Input line */
float*              sample      /* Input sample */
)
{
  float             i;                  /* Intermediate X coordinate */
  float             j;                  /* Intermediate Y coordinate */
  int               b;                  /* Intermediate block */
  float             l;                  /* Intermediate line */
  float             s;                  /* Intermediate sample */
  char              msg[STRLEN];        /* Warning message */

/* Compute intermediate coordinates */

  i = (float)((x - xc) / sx);
  j = (float)((y - yc) / sy);

/* Check for very small numbers in i and j and assume they are zero */

  i = (fabs(i) < 1E-5 ? 0.0 : i);
  j = (fabs(j) < 1E-5 ? 0.0 : j);

/* Compute block and check range */

  b = (int)(floor((i + 0.5) / nl)) + 1;
  if (b < 1 || b > nb) {
    sprintf(msg, "block is out of range (1 < %d < %d)", b, nb);
    WRN_LOG_JUMP(msg);
  }

/* Compute line and check range */

  l = (float)(i - ((b - 1) * nl));
  if (l < -0.5 || l > nl - 0.5) {
    sprintf(msg, "line is out of range (0 < %e < %d)", l, nl);
    WRN_LOG_JUMP(msg);
  }

/* Compute sample and check range */

  s = (float)(j - absOffset[b-1]);
  if (s < -0.5 || s > ns - 0.5) {
    sprintf(msg, "sample is out of range (0 < %e < %d)", s, ns);
    WRN_LOG_JUMP(msg);
  }

/* Set return values */

  *block = b;
  *line = l;
  *sample = s;

  return(0);

 ERROR_HANDLE:

  *block = -1;
  *line = -1.0;
  *sample = -1.0;
  return(1);
}
```

4.7.4 misrinv.c

```
#include “misrproj.h”                    /* Prototype for this function */
#include “errormacros.h”          /* Error macros */
extern int nb;
extern int nl;
extern int ns;
extern float absOffset[NBLOCK];
extern double ulc[2];
extern double lrc[2];
extern double sx;
extern double sy;
extern double xc;
extern double yc;

#define FUNC_NAMEm “misrinv”

int misrinv(
const int     block,     /* Input block */
const float             line,       /* Input line */
const float             sample,   /* Input sample */
double*                 x,                  /* Output SOM X coordinate */
double*                 y                   /* Output SOM Y coordinate */
)
{
 int                    n;                  /* Number of line to current block */
 char                   msg[STRLEN];     /* Warning message */

/* Check Arguments */

 if (block < 1 || block > NBLOCK) {
  sprintf(msg, “block is out of range (0 < %d < %d)”, block, nb);
  WRN_LOG_JUMP(msg);
 }

 if (line < -0.5 || line > nl - 0.5) {
  sprintf(msg, “line is out of range (0 < %e < %d)”, line, nl);
  WRN_LOG_JUMP(msg);
 }

 if (sample < -0.5 || sample > ns - 0.5) {
  sprintf(msg, “sample is out of range (0 < %e < %d)”, sample, ns);
  WRN_LOG_JUMP(msg);
 }

/* Compute SOM x/y coordinates in ulc/lrc units (meters) */

 n = (int)((block - 1) * nl * sx);
 *x = (double)(xc + n + (line * sx));
 *y = (double)(yc + ((sample + absOffset[block-1]) * sy));

 return(0);

ERROR_HANDLE:
```

```
  *x = -1e-9;
  *y = -1e-9;
  return(1);
}
```

4.7.5 misrproj.h

```
#ifndef MISRPROJ_H
#define MISRPROJ_H
/* Defines */

#define STRLEN      200
#define NBLOCK      180
#define NOFFSET     NBLOCK - 1
#define R2D         57.2957795131
#define D2R         1.745329251994328e-2
#define NPROJ       13

/* Prototypes */

int misr_init(
const int     nblock,       /* Number of blocks */
const int         nline,        /* Number of lines in a block */
const int         nsample,      /* Number of samples in a block */
const float           relOff[NOFFSET],/* Block offsets */
const double          ulc_coord[],    /* Upper left corner coord. in meters */
const double          lrc_coord[]     /* Lower right corner coord. in meters */
);

int misrfor(
const double          x,                  /* Output SOM X coordinate */
const double          y,                  /* Output SOM Y coordinate */
int*                  block,      /* Input block */
float*                line,       /* Input line */
float*                sample      /* Input sample */
);

int misrinv(
const int     block,      /* Input block */
const float           line,       /* Input line */
const float           sample,     /* Input sample */
double*               x,                  /* Output SOM X coordinate */
double*               y                   /* Output SOM Y coordinate */
);

#endif /* MISRPROJ_H */
```

4.7.6 errormacros.h

```
#ifndef ERRORMACROS_H
#define ERRORMACROS_H
```

```
#include <stdio.h>

#define HDFEOS_ERROR_CHECK(msg) \
  if (hdfeos_status_code == FAIL) { \
    fprintf(stderr, "Error: %s at line %d\n", msg, __LINE__); \
    exit(1); \
  }

#define MEM_ERROR_CHECK(msg) \
  if (mem_status_code == NULL) { \
    fprintf(stderr, "Error: %s at line %d\n", msg, __LINE__); \
    exit(1); \
  }

#define ERROR(msg) \
  { \
    fprintf(stderr, "Error: %s at line %d\n", msg, __LINE__); \
    exit(1); \
  }

#ifdef MISRWARN
  #define WRN_LOG_JUMP(msg) \
   { \
     fprintf(stderr,"Warning: %s in %s <Line: %d>\n", \
             msg, FUNC_NAMEm, __LINE__); \
     goto ERROR_HANDLE; \
   }
#else
  #define WRN_LOG_JUMP(msg) goto ERROR_HANDLE;
#endif

#endif /* ERRORMACROS_H */
```

4.7.7 Makefile

```
CC=     gcc
#CFLAGS= -g -n32 -DMISRWARN -I$(HDFINC) -I$(HDFEOS_INC)
CFLAGS= -g -n32 -I$(HDFINC) -I$(HDFEOS_INC)
LDFLAGS=-L$(HDFEOS_LIB) -L$(HDFLIB) \
       -lhdfeos -lGctp -lmfhdf -ldf -ljpeg -lz –lm

OBJS=   misr_init.o \
        misrinv.o \
        misrfor.o

all: misrcoordex

misrcoordex: misrcoordex.o $(OBJS)
        $(CC) $(CFLAGS) -o $@ $@.o $(OBJS) $(LDFLAGS)

clean:
        /bin/rm -f misrcoordex misrcoordex.o $(OBJS)
```

misrcoordex.o $OBJS: misrproj.h errormacros.h

4.8 Bibliography

[1] Bull, Mike, et. al., MISR Data Products Specifications -- Incorporating the Science Data Processing Interface Control Document, aka the MISR DPS, JPL D-13963, Revision S, April, 2011. (http://eosweb.larc.nasa.gov/PRODOCS/misr/dps/)

[2] Snyder, John, Map Projections--A Working Manual, USGS Professional Paper 1395, 1987, Ch. 27, "Space Oblique Mercator Projection", pp. 214-229.

[3] U.S. Geological Survey, National Mapping Division, GCTP General Cartographic Transformation Package Software Documentation, 1993.

[4] Klein, Larry, HDF-EOS Library Users Guide for the ECS Project, Volume 2: Function Reference Guide, 170-TP-501-001, June 1999, Raytheon Systems Company, Upper Marlboro, MD, pages 1-7 through 1-12 and 2-152. (http://edhs1.gsfc.nasa.gov/waisdata/sdp/pdf/tp17050101.pdf)

5 ACRONYM LIST

ASDC.....Atmospheric Sciences Data Center (see LaRC DAAC)
BRF.....Bidirectional Reflectance Factor
DAAC.....Distributed Active Archive Center
ECS.....EOSDIS Core System (Data Production System at DAAC)
EOS.....Earth Observing System
EOSDIS.....Earth Observing System Data and Information System
ESDT.....Earth Science Datatype
JPL.....Jet Propulsion Laboratory
LaRC DAAC.....NASA Langley Reserach Center DAAC
NASA.....National Aeronautics and Space Administration
SCF.....Science Computing Facility
SDP.....Science Data Processing
SOM.....Space-Oblique Mercator
WGS84.....World Geodetic System 1984